ITERATIVE ALGORITHMS FOR VIRTUAL MACHINE PLACEMENT IN CLOUD ENVIRONMENTS

BY

ABUBAKAR BALA

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

OCTOBER, 2014.

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **ABUBAKAR BALA** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

Dr. Sadiq M. Sait (Adviser)

Dr. Aiman H. El-Maleh (Member)

Dr. Ahmad Almulhem (Member)

Dr. Ahmad Almulhem
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

8/2/15
Date

ITERATIVE ALGORITHMS FOR VIRTUAL MACHINE PLACEMENT IN CLOUD ENVIRONMENTS

by

ABUBAKAR BALA

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the degree

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

KING FAHD UNIVERSITY
OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

OCTOBER 2014

Dedication

I give all thanks and gratitude to Almighty Allah for given me the life, strength and health to complete this thesis work. My special thanks also goes to my loving parents Hafsat Yusuf Garba and Yusha'u Bala Falke who took the onerous task of educating us right from childhood. Moreover, I will also like to extend my sincere appreciation to my siblings especially my beloved sister Aisha Yusha'u Bala and her Husband Baba Yusuf Abubakar. The moral and financial support of my maternal grandfather Yusuf Garba Ali is highly acknowledged and appreciated. My gratitude also goes to my beloved fiance Ummi Nuruddeen Abdul-Kadir whom I have not met physically till now- I really appreciate your patience. Also to my late Uncle Usman Yusuf, your advice and counseling is really appreciated- rest in peace. To my best friends at KFUPM, Abba A. Abubakar and Dahiru U. lawal I say a big 'thank you' to you two for making life at KFUPM a very lively one. To all members of the Nigerian community at KFUPM, especially the JALSA committee members and the team members of Nigerian United Football Club (NUFC), I extend my sincere gratitude to you all for making my life at KFUPM a very memorable one. Finally, I would also like to use this opportunity to thank all my school teachers at: Corona primary school Bukuru, ST. Murumba College Jos, ST. Thomas secondary school Kano, Bayero University Kano and King Fahd University of Petroleum & Minerals.

ACKNOWLEDGMENTS

I acknowledge the academic and moral support of my thesis advisor, Dr. Sadiq M. Sait who did not only guide me as a student, but as a father will do to his own child. Furthermore, I also thank Dr. Aiman El-Maleh for introducing me to cuckoo search optimization (CSO) algorithm as a course project. In addition, I would like to extend my gratitude to Dr. Ahmad AlMulhem for accepting to be a member of my thesis committee. I also acknowledge Mr. Khawaja S. Shahzada for sharing information regarding the virtual machine placement problem. Additionally, I extend my gratitude to Dr. Hamza O. Salami for offering me free tutorials on MATLAB. Moreover, I acknowledge the support of my place of work Bayero University Kano (BUK) for paying my salary throughout my Msc program at KFUPM. Finally, I extend my sincere appreciation to King Fahd University of Petroleum & Minerals for offering me a scholarship to pursue my Msc.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: Abubakar Bala

TITLE OF STUDY: ITERATIVE ALGORITHMS FOR VIRTUAL MACHINE PLACEMENT IN CLOUD ENVIRONMENTS

MAJOR FIELD: Computer Engineering

DATE OF DEGREE: October 2014

Virtualization technology has facilitated the placement of a large number of independent virtual machines (VMs) on a single physical server. The virtual machine placement problem is that of mapping these VMs onto physical machines while attempting to optimize certain design objectives. This has become a very challenging task especially in datacenters that receive many VM requests. This thesis is divided into two parts. In the first part, we engineer the cuckoo search optimization (CSO) algorithm, a novel nature-inspired population-based metaheuristic algorithm, to solve the server consolidation problem of datacenters. Moreover, we use a new fitness measure to determine the quality of placement solutions. Experiments we conducted show that the CSO algorithm for server consolidation is able to outperform the Reordered Grouping Genetic Algorithm (RGGA) and Grouping

Genetic Algorithm (GGA). In the second part of this thesis, the VM placement problem is formulated as a multiobjective optimization problem, aiming at the simultaneous minimization of power consumption and resource wastage of data-centers. In this part we designed and implemented a multi-objective cuckoo search optimization (CSO) algorithm to place VMs on datacenters while simultaneously optimizing the power consumption and resource wastage of the datacenter. The And-Like-Fuzzy-Aggregation (AFA) multi-objective evaluation function is used as a fitness measure to combine the power consumption and resource wastage objectives. Experimental results obtained demonstrate that the multi-objective CSO algorithm clearly outperforms the Reordered Grouping Genetic Algorithm (RGGA) and Grouping Genetic Algorithm (GGA).

ملخص

الإ سم: ابوبكر بالا

العنوان: خوارزميات تكرارية الظاهري ل وضع الجهاز في البيئات السحابية

التخصص رئيسي : هندسة الكمبيوتر

تاريخ الدرجة: ذو الحجة ١٤٣٥ - تشرين الأول ٢٠١٤

تقنية التمثيل الافتراضي سهلت تنسيب عدد كبير من الأجهزة الظاهرية المستقلة على خادم فعلي واحد. مشكلة تنسيب الأجهزة الظاهرية هي عبارة عن تعيين الأجهزة الظاهرية على الأجهزة الفعلية مع محاولة تحسين بعض أهداف التصميم. أصبحت هذه مهمة صعبة جدا وخصوصا في مراكز البيانات التي تتلقى العديد من طلبات الأجهزة الظاهرية. تنقسم هذه الأطروحة إلى قسمين. في الجزء الأول، فإننا نهندس خوارزمية بحث الوقواق الأمثل المستوحاة من الطبيعة لحل مشكلة دمج الخوادم في مراكز البيانات. وعلاوة على ذلك، نستخدم مقياس لياقة جديد لتحديد جودة حلول التنسيب. التجارب التي أجريناها تظهر أن خوارزمية بحث الوقواق الأمثل لدمج الخوادم قادرة على التفوق على الخوارزميات الجينية التجمع. وعلاوة على ذلك، المقارنة بين خوارزمية بحث الوقواق الأمثل مع إصدارات محسنة من الارشاديات المبنية على المناسب الأول المخفض والأقل حمولة تشير إلى أنها أفضل وقادرة على العثور على المواضع مع عدد أقل من الخوادم المادية في غضون فترة زمنية حسابية تنافسية. في الجزء الثاني من هذه الأطروحة، تم صياغة مشكلة تنسيب الأجهزة الافتراضية كمشكلة تحسين متعددة الأهداف، تهدف إلى التقليل من استهلاك الطاقة وهدر الموارد من مراكز البيانات في وقت واحد. يتم استخدام دالة التجميع الضبابية كمقياس لياقة للجمع بين أهداف استهلاك الطاقة وموارد الهدر. النتائج التجريبية التي تم الحصول عليها تبين أن خوارزمية بحث الوقواق الأمثل متعددة الأهداف تتفوق بوضوح على الخوارزمية الجينية التجمع بالاضافة الى الطرق المبنية على المناسب الأول المخفض والأقل حمولة.

CHAPTER 1

INTRODUCTION

In this chapter we present an introduction to the virtual machine placement problem. Moreover, the chapter introduces the novel cuckoo search optimization (CSO) algorithm. Additionally, it provides information regarding: aims and objectives of the study, scope of the research study, methodology adopted in the study as well as an outline of this thesis report.

1.1 Thesis Overview

Due to the recent increase in the number of clients that subscribe to cloud services, datacenters are now faced with the problem of managing large number of virtual machines. The virtual machine placement problem is that of placing these virtual machines (VMs) onto the datacenter while attempting to optimize specific design objectives like: thermal dissipation, power consumption, resource wastage and number of physical machines used for placement. This research study is divided into two parts. In the first part reported in Chapter 3, the CSO algorithm

is designed to solve the VM placement problem while targeting the minimization of number of physical machines used for placement (also called server consolidation). In contrast, the second study reported in Chapter 4, aims at simultaneously optimizing the datacenter for reduced power consumption and resource wastage in what is widely known as multi-objective optimization.

1.2 Virtual Machine Placement Problem

This section presents details of the virtual machine placement problem. Moreover, it outlines the two level control architecture for automatic management of virtual machine placements in datacenters.

Recently, datacenters have become more robust. This is due to the employment of virtualization technology which enables the resources of a single server to be divided into several isolated execution environments running as virtual machines. This has resulted in the creation of datacenters with fewer physical servers, high per-server utilization, higher availability, enhanced flexibility, as well as reduced hardware and operational costs. However, this flexibility provided by virtualization poses new research challenges. Provisioning and management of large number of virtual machines have remained a very challenging task especially in large datacenters. This thesis considers virtualized datacenters that provide clients with a shared hosting infrastructure to run their applications on a virtualized platform. Thus, each application runs on its own virtual machine which can be managed and provisioned on-demand. Clients usually place requests for resources and it is

the responsibility of the datacenter manager to find placements for these virtual machines (VMs). Moreover, the manager has to also determine the amount of resource that will be allocated to each VM. This task is often difficult and can hardly be handled by humans, especially in the case of large datacenters with thousands of daily VM requests [1].

Virtual machine placement in datacenters can be formulated as a vector bin-packing problem. In such problems, there are items of different sizes, which have to be packed into bins such that the minimum numbers of bins are used, within the given capacity of each bin. Figure 1.1 shows a packing of three items into a single bin. In the case of the VM placement problem, items are represented as VMs, bins as servers and the dimensions are the physical resources (e.g. CPU and memory as in Figure 1.1).

The bin packing problem is an old problem with a variety of applications such as: material cutting, scheduling, loading and layout design. However, even the simple one-dimensional bin packing problem is known to be NP-hard [2].

A two-level control approach (shown in Figure 1.2) for automating the management of resources in datacenters was developed by *Tolia et al.* [3]. The local controller in Figure 1.2 is responsible for estimating the amount of compute resources required by applications to guarantee their performances. This estimation usually involves either some form of approximation or profiling- in which an application is run on a server for few weeks and then the peak utilization of resources are taken as the resource utilization request for such application. In general, the

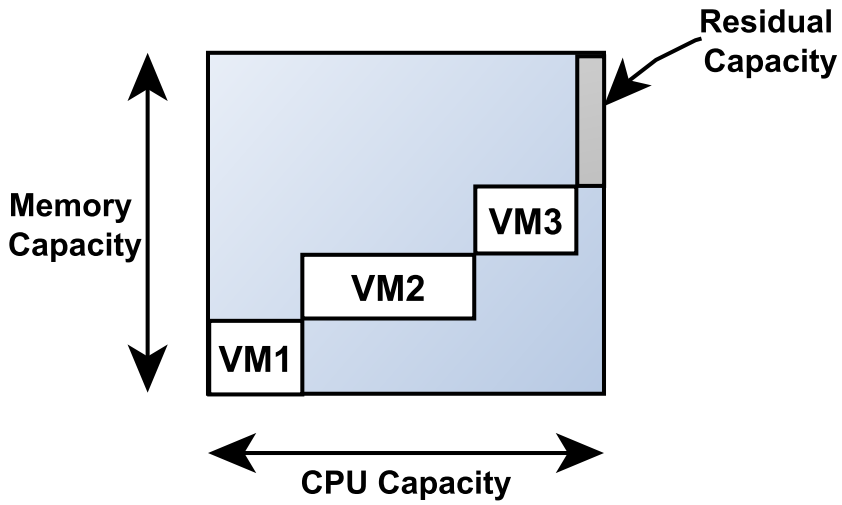


Figure 1.1: A typical example of VM placement on a single server.

local controller maps applications to physical resource requirements. On the other hand, the global controller (shown in Figure 1.2) is in charge of final VM placement and resource allocation. The global controller receives resource utilization requests in the form of VM requests from the local controller and then finds the placement and amount of physical resource to allot to each VM.

In the VM placement problem, the job of the global controller is to determine placements for the pooled VM requests it receives from the local controller. This placement is usually done with the aim of optimizing the datacenter for certain design objectives such as: power consumption, resource wastage and number of physical servers used for placement. However, this initial placement carried-out by the global controller may have to change over time. This is due to the fact that workloads represented as VMs are dynamic in nature. This means that some VMs may change their resource requirements or release resources due to task completion. Moreover, additional VM requests may enter into the datacenter. In each of the aforementioned cases, the global controller should be able to receive some feedback on the condition of workloads within the datacenter. From the Feed-Back block (shown in Figure 1.2), the global controller is capable of finding better placements for existing as well as incoming VM requests. VM migration technology is often used to aid the global controller achieving the dynamic placement of VMs.

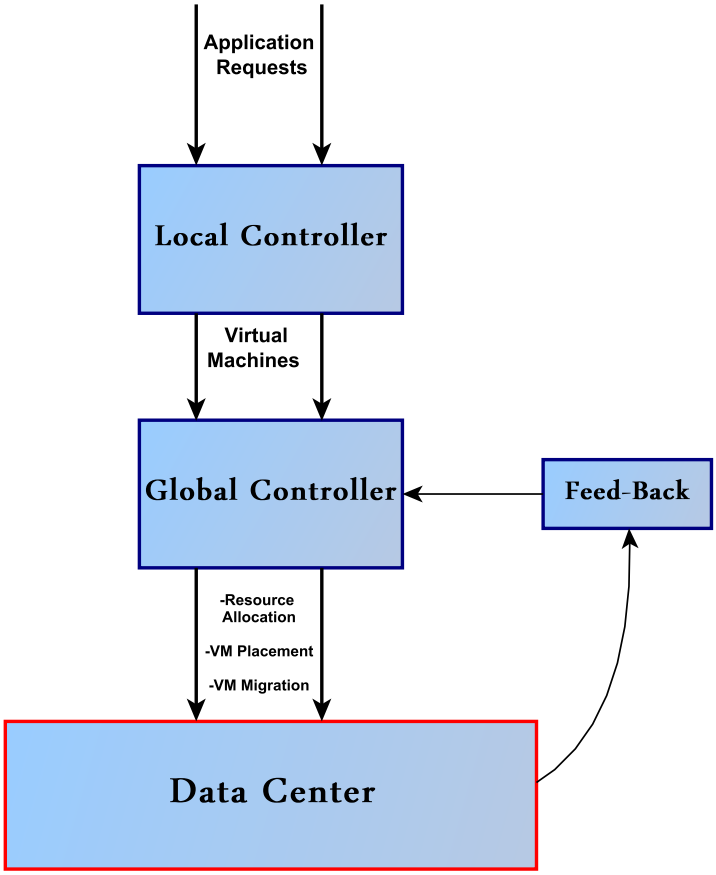


Figure 1.2: Two-level control architecture for automatic resource management in datacenters.

1.3 Cuckoo Search Optimization Algorithm

In this section we present an introduction to the cuckoo search optimization (CSO) algorithm. The cuckoo search optimization (CSO) algorithm (shown in Algorithm 1) is inspired from the aggressive reproduction strategy of the wonderful cuckoo bird [4]. Some species of cuckoos engage in obligate brood parasitism behaviour-in which a bird (the parasite) lays its egg in the nest of another bird (the host) so that the host becomes responsible for the incubation and hatching of the parasites' egg(s). However, some of these host birds are very vigilant and could engage the intruding bird cuckoo in direct conflict. Moreover, if the host discovers that some of the eggs in its nest are not its own, it either throws the alien ones away or discards the entire nest and builds a new one elsewhere. Other species of cuckoo such as New World Brood-Parasitic *Tapera* have evolved in such a way that they lay eggs that mimic the color and size of that of their hosts. This reduces that chance that their eggs will be identified and eventually discarded, thus increasing their reproductivity. Most parasitic cuckoos lay their eggs in fresh host nests. This increases the chance that their eggs will get hatched earlier than that of their hosts. Once the first cuckoo bird is hatched, it blindly propels other eggs out of the nest to increase its share of food provided by the host bird. Walton *et al.* [5] presented two modifications to the original cuckoo search by Yang *et al.* [4]. These improvements enable the cuckoo search achieve faster convergence rate as well as wider application. The modified cuckoo search optimization algorithm is shown in Algorithm 2.

The CSO algorithm employs the use of Lévy flight for both local and global searching. The lévy flight is a random walk characterized by sudden jumps. Studies have shown that such characteristic is demonstrated by the flight behaviour of many insects and animals. Recent applications of such behaviour in optimization and optimal search has yielded interesting results [6]. Figure 1.3 shows a typical plot of the Lévy flight.

1.3.1 Approximating the Lévy Flight

In this work, we make use of an approximation of the Lévy process in order to generate random increments that are drawn from Lévy flight. The Lévy flight step increment/length conforms to the power law distribution [7]. To transform a uniformly distributed random variable into another distribution we need to find the inverse cumulative distribution function. For example if F is a cumulative distribution function corresponding to the probability density f, and u is a uniform random variable in the range [0,1], then: $x = F^{-1}(u)$ is distributed according to F.

The cumulative distribution function for a typical a pure power law distribution is:

$$F(x) = 1 - (x/x_{min})^{-\alpha} \tag{1.1}$$

Where x_{min} is the minimum value that the random variable can take and α is

the step size whose value depends on the scale of the problem of interest. α can be made fixed as in the case of original cuckoo search algorithm or made to evolve with increase in generation in the case of the modified cuckoo search algorithm. Hence, the inverse distribution function becomes;

$$F^{-1}(u) = x_{min}(1-u)^{-1/\alpha} \tag{1.2}$$

Thus from Equation 1.2 we can conveniently create random variables that follow the power law distribution of Equation 1.2. Thus using this equation we can come up with random step increments that are drawn from Lévy flight.

Algorithm 1 Original CSO Algorithm

Initialize a population of n host nests $X_i = 1, 2 \ldots, n$
for all X_i **do**
    Calculate fitness $F_i = f(X_i)$
end for
Set $MAXiter$ {$MAXiter$ is the maximum iteration}
Generation number $G \leftarrow 1$
for $G = 1$ **to** $MAXiter$ **do**
    Generate cuckoo egg (X_j) by taking Lévy flight from random nest
    $F_j = f(X_j)$
    Choose a random nest i
    if $F_j > F_i$ **then**
        $X_i \leftarrow X_j$
        $F_i \leftarrow F_j$
    end if
    Abandon a fraction P_a of worst nests
    Build new nests at new locations by Lévy flight to replace abandoned nests
    Evaluate fitness of new nests and rank entire solutions
end for

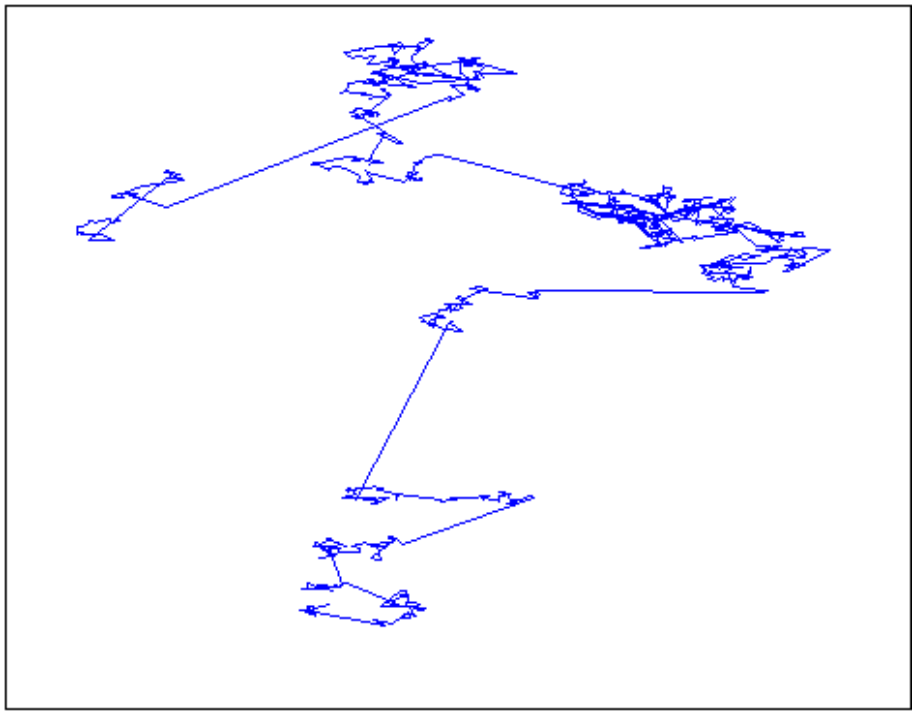


Figure 1.3: A typical plot of Lévy flight.

Algorithm 2 Modified Cuckoo Search Algorithm

$A \leftarrow 1$ {Maximum Lévy flight step size}
$\phi \leftarrow GoldenRatio$
Set $MAXiter$ {$MAXiter$ is the maximum iteration}
Initialize the Population
for all X_i **do**
    Calculate fitness $F_i = f(X_i)$
end for
Generation number $G \leftarrow 1$
for $G = 1$ **to** $MAXiter$ **do**
    Sort nests according to fitness
    partition the population into top and bottom nests

---

    for all X_i such that X_i is in bottom nests **do**
        Calculate Lévy flight step size $\alpha \leftarrow A/\sqrt{G}$
        Perform Lévy from X_i to create a new nest X_j
        $X_i \leftarrow X_j$
        $f(X_i) \leftarrow f(X_j)$
    end for

---

    for all X_i such that X_i is in top nests **do**
        Select another random nest from the top nests X_j
        if $X_i = X_j$ **then**
            Calculate Lévy flight step size $\alpha \leftarrow A/G^2$
            Perform Lévy flight from X_i to create a new nest X_k
            Select a random nest X_l from the entire population
            if $f(X_k) > f(X_l)$ **then**
                $X_l \leftarrow X_k$
                $f(X_l) \leftarrow f(X_k)$
            end if

---

        else
            Move a distance $d_x = |X_i - X_j|/\phi$ from worse nest to the better nest to find X_k
            Select a random nest X_l from the entire population
            if $f(X_k) > f(X_l)$ **then**
                $X_l \leftarrow X_k$
                $f(X_l) \leftarrow f(X_k)$
            end if
        end if
    end for
end for

1.4 Aims of Study

This thesis has two main aims:

- The implementation of cuckoo search optimization (CSO) algorithm to minimize the number of physical machines used for placement in the datacenter.
- The implementation of a multi-objective cuckoo search optimization (CSO) algorithm to simultaneously minimize the power consumption and resource wastage of the datacenter.

1.5 Methodology

The following is the methodology followed in this research work:

1. The generation of a dataset which will represent VM placement requests.
2. Design and implementation of the CSO algorithm for server consolidation in datacenters.
3. Design and implementation of the multi-objective CSO for simultaneous optimization of the power consumption and resource wastage of datacenters.
4. Implementation of other VM placement techniques:Reordered Grouping Genetic Algorithm (RGGA) and Grouping Genetic Algorithm (GGA).
5. Testing the performance of CSO against other existing VM placement techniques

6. Testing the two developed CSO algorithms for scalability.

7. Providing a comprehensive discussion of the implications of comparison results obtained and how the CSO scales with increase in number of VM requests.

1.6 Scope and Limitations

- This study assumes that the local controller has already mapped each application request to its resources utilization requests. Thus the focus of this work is the design of global controller.

- This research assumes a homogeneous datacenter i.e. the servers in the datacenter are all identical.

- A virtual machine request is characterized by two dimensions: The CPU and memory dimensions. Disk dimension is not considered because we assume that network-attached storage (NAS[1]) is used as the main storage.

1.7 Thesis Outline

The remaining part of this thesis report is organized as follows: Chapter 2 reviews recent published literatures on VM placement Chapter 3 provides the methodology as well as experimental results obtained from the employment of CSO algorithm to solve the server consolidation problem in datacenters. Moreover, the chapter

[1]NAS device is a dedicated server used for storing and sharing files.

provides discussions on the performance of CSO against the RGGA, and GGA placement techniques. Furthermore, in Chapter 4, methodology and experimental results obtained from the use of CSO algorithm for multiobjective optimization of datacenters is presented. Additionally, the chapter discusses how the multiobjective CSO algorithm performed compared to the RGGA and GGA algorithms. Finally, in Chapter 6 we discuss the conclusions that can be drawn from the application of CSO algorithm in Chapter 3 and 4. Finally, the chapter presents future work and possible extensions that can be done to this research work.

1.8 Chapter Summary

In this chapter we have presented general background knowledge on the virtual machine placement problem. Moreover, we have introduced the cuckoo search optimization (CSO) algorithm. The chapter also presents: the aims of the research work, methodology of the research, as well as scope and limitations of the study. In the next chapter, we present a review of recent VM placement techniques.

CHAPTER 2

LITERATURE REVIEW

In this chapter we present a review of recent literature related to the virtual machine placement problem. Furthermore, this chapter broadly categorizes VM placement techniques into deterministic and non-deterministic methods.

2.1 Deterministic Methods

Since the VM placement problem is a variant of the popular vector bin-packing problem, deterministic/exact methods used for solving the bin packing have been modified to solve the VM placement problem.

The First-Fit-Decreasing (FFD) heuristic and Least-Loaded (LL) heuristic [8] are among the prominent of these methods. The FFD algorithm is shown in Algorithm 3. In the algorithm, VMs to be packed are initially sorted in decreasing order of their sizes. In single dimensional bin-packing, this sorting process is trivial. However, in the case of multidimensional bin-packing, Maruyama *et al.* [8] have proposed eight different methods for sorting the VMs according to their

multidimensional sizes. Subsequently, each VM in the sorted list is then packed in the first existing server that can accommodate it. The function $Packable(x_i, S_j)$ in the FFD algorithm returns true if VM x_i is packable into server S_j, otherwise it returns false. If all the existing servers are sequentially scanned and none could accommodate a VM, new server is introduced into the solution to accommodate such VM (Algorithm 3 line 12 and 13) and the algorithm keeps packing remaining VMs till all the VMs have been successfully packed.

In contrast, the LL algorithm (Shown in Algorithm 4) tries to balance loads among the existing servers by assigning a VM to the least loaded bin. There are two main distinctions between the FFD and the LL heuristics. Firstly, in the LL algorithm, each time a new server is introduced into the solution, a repacking of VMs is done (Algorithm 4 line 17). This is aimed at balancing the loads among the currently existing servers. Secondly, in the LL algorithm, before a VM is packed, all the existing servers are sorted in ascending order of utilization and the VM is placed at the topmost server (least-loaded server) Algorithm 4 line 8.

Ajiro et al. [9] proposed a new technique for improving the standard FFD and LL. Experiments they conducted proved that their improved versions of FFD and LL outperform the standard FFD and LL algorithms. In the improved FFD and LL, when VMs fail to be packed into any of the existing servers, the VMs to be packed are reordered in such a way that the VMs that fail to pack are now packed first. This reordering process is repeated $MAXR$ time. However, if after $MAXR$ times the VMs can still not be packed into the existing servers, a new server is

introduced into the solution.

Other variants of deterministic methods for VM placement include the Best-Fit-Decreasing (BFD) heuristic [1]. The BFD is similar to the FFD in all aspects, except that in the BFD, a VM is placed in the fullest server that still has enough space to accommodate it. A similar method is the Next-Fit (NF) heuristic, in which only one server is considered at any particular time, when no VM can be placed in the server, it is then 'shipped' away and a new empty server is brought into the solution. Thus, the NF is a sequential algorithm because it processes both VMs and servers sequentially.

Algorithm 3 FFD Algorithm

```
1: {x_1, x_2, ..., x_n} ← Sort(VMs) {Sort VMs in descending order of sizes}
2: m ← 1 {Set number of servers to 1}
3: S_1 ← {} {Add an empty server}
4: for i ← 1 to n do
5:     for j ← 1 to m do
6:         if Packable(x_i, S_j) then
7:             S_j ← S_j ∪ {x_i}
8:             break
9:         end if
10:    end for
11:    if j == m and Packable(x_i, S_m) == False then
12:        m ← m + 1 {Add a new server}
13:        S_m ← {x_i} {Place VM x_i in new server}
14:    end if
15: end for
```

2.2 Non-Deterministic Methods

In contrast, other methods for solving the VM placement problem are non-deterministic in nature. These methods have been shown to outperform most deterministic methods for VM placement [10]. One of the well known of these techniques is the Grouping Genetic Algorithm (GGA) (Shown in Algorithm 5) of Falkenauer [11]. The GGA was developed to address the failure of the classic genetic algorithm (GA) in solving most grouping problems. It employs the use of a group-based encoding scheme rather than the individual or item-based encoding of the classic GA. However, due to this new encoding scheme, Falkenauer [11] designed new mutation and crossover operations that apply to the new encoding. An additional work is that of Feller *et al.* [12] where the VM placement problem is modeled as an instance of multi-dimensional vector bin-packing problem. In addition, a modified ant colony optimization (ACO) algorithm is employed to solve the server consolidation problem in datacenters. The ACO-based algorithm found placement solutions that employ less number of physical servers and less power consumption than the FFD heuristic. In a related work, Gao *et al.* [10] also designed an ACO-based system for solving the VM placement problem in datacenters called VMPACS. They formulated the VM placement problem as a multi-objective combinatorial optimization problem with the objective of simultaneously optimizing the power consumption and total resource wastage of datacenters. Furthermore, Xu *et al.* [1] also formulated the VM placement problem as a multi-objective optimization problem, simultaneously minimizing power

Algorithm 4 LL Algorithm

```
1: {x_1, x_2, ..., x_n} ← Sort(VMs) {Sort VMs in descending order of sizes}
2: m ← LB({x_1, x_2, ... x_n}) {Set number of servers to the lower bound}
3: while true do
4:     for j ← 1 to m do
5:         S_j ← {} {Adding an empty server}
6:     end for
7:     for i ← 1 to n do
8:         {S_1, S_2, ..., S_m} ← Sort(servers) {Sort servers in ascending order}
9:         for j ← 1 to m do
10:            if Packable(x_i, S_j) then
11:                S_j ← S_j ∪ {x_i}
12:                break
13:            end if
14:        end for
15:        if j == m and Packable(x_i, S_m) == False then
16:            m ← m + 1 {Add new server}
17:            break {Do repacking}
18:        end if
19:    end for
20:    if i == n then {If all VMs are placed into servers}
21:        break {End algorithm}
22:    end if
23: end while
```

consumption, total resource wastage and thermal dissipation costs of datacenters. They employed a modified grouping genetic algorithm (MGGA) with Fuzzy multi-objective evaluation to effectively search the large design space.

The CSO-based [4] VM placement algorithm is also a non-deterministic algorithm. In this thesis, we employ the CSO algorithm to optimize the placement of VMs onto datacenters. We divide the experiments into two chapters. In Chapter 3 we employ the CSO algorithm to optimize the datacenter for the minimization of the number of physical machines used -server consolidation . In Chapter 4 we present findings on the employment of the CSO algorithm to optimize the datacenter for reduced power consumption and resource wastage -multi-objective optimization. The CSO algorithm is a novel nature-inspired algorithm with wide applications [5, 13, 14]. One of the distinctive features of the CSO is the employment of Lévy flight which enables the algorithm to traverse the large design space of most combinatorial optimization problems. Additionally, the CSO has less tuning parameters than most other meta-heuristic algorithms such as ACO, PSO and GA [4]. To the best of our knowledge this is the first application of CSO algorithm to solve the VM placement problem in datacenters.

2.3 Chapter Summary

This chapter presented a review of existing VM placement techniques. In the next chapter we discuss the first experiment which is the design of the CSO algorithm for solving the server consolidation problem in datacenters.

Algorithm 5 GGA Algorithm

Set $MAXiter$ {$MAXiter$ is the maximum iteration}
Set X_r {X_r is the crossover rate}
Set M_r {M_r is the mutation rate}
Set C {C is the population size}
Generation number $G \leftarrow 1$
$P \leftarrow Initialize(C)$ {Initialize the population}
for $i \leftarrow 1$ **to** $MAXiter$ **do**
    for $j \leftarrow 1$ **to** $C \times X_r$ **do**
        $(x, y) \leftarrow Select\,(P)$ {select parents x and y by roulette wheel}
        offspring$[j, j+1] = Crossover\,(x, y)$
    end for
    for $j \leftarrow 1$ **to** $C \times M_r$ **do**
        $z = Select\,(P)$ {select a random parent z}
        offspring$[j + (C \times X_r + 1)] \leftarrow Mutation\,(z)$
        $P \leftarrow EvaluateSelect(P$,offspring) {evaluate and choose the best C chromosomes}
    end for
    $P \leftarrow Inversion\,(P)$ {reshuffle each chromosome}
end for

CHAPTER 3

CSO FOR SERVER CONSOLIDATION

3.1 Introduction

This chapter looks into the design of the cuckoo search optimization (CSO) algorithm for solving the server consolidation problem of datacenters. In server consolidation problem, it is required that VM requests from clients are placed onto the datacenter in such a way that a minimum number of physical servers are used. As stated earlier, with server consolidation, idle or unnecessary servers in the datacenter can be turned off/suspended [12]. This has the potential effect of cutting down the energy consumed and thermal dissipation of datacenters. Thus, providing cost savings for both the cloud service provider as well clients. Moreover, in this chapter we employ the use of the fitness measure proposed by Sadiq *et al.* [15] to determine the quality of solution in the population pool of the CSO

algorithm. Experimental results obtained show that improvement of this fitness measure correlates with the minimization of number of physical servers used for placement. In addition, results at the end of the chapter show that the CSO algorithm outperforms other server consolidation methods such as: Re-ordered Grouping Genetic Algorithm (RGGA) and Grouping Genetic Algorithm (GGA).

3.2 Problem Definition

This section details the VM placement optimization problem and also elaborates on the optimization equation and constraints. In this work, the datacenter is considered to be fully virtualized such that all applications run on virtual machines. Thus the virtual machine placement problem is that of assigning these VMs to physical machines such that certain design objectivesare optimized (in this case number of servers used for placement). Thus the problem is a variant of multi-dimensional vector bin-packing problem with dimensions as resource utilizations. CPU and memory dimensions are used to characterize a VM and a server node. If a server hosts more than one VM, the CPU utilization of the server is estimated as the sum of CPU utilizations of the VMs it hosts. Similarly, the memory utilization of the server is approximated as the sum of memory utilizations of the VMs. However, to prevent the CPU and memory utilizations of a server from reaching 100%, some threshold value is set as upper bound on resource utilization. The main reason for this threshold is that 100% utilization of resources on a server can cause huge performance degradation and may limit the use of VM-migration.

3.2.1 Optimization formulation

In this section, the VM placement equation and constraints are defined. Suppose there are n number of VMs that can be placed on m servers. Such that no VM has resource request that cannot be handled by a single server. Let ρ_i^c denote CPU request of VM_i and ρ_i^m the memory request of VM_i. Additionally, let T_{cj} and T_{mj} be the threshold of CPU and memory utilization of server j respectively. In addition we define the following two binary decision variables:

- Server allocation variable y_j, equals 1 if server j is in use and 0 otherwise.
- VM allocation variable $x_{i,j}$, equals 1 if VM i is placed in server j, and 0 otherwise.

Since the fundamental aim of the consolidation algorithm is to place VMs such that the minimum number of servers are used, the placement problem can be formulated as:

$$\text{Minimize } f(y) = \sum_{j=1}^{m} y_j$$

subject to:

$$\sum_{i=1}^{n} \rho_i^c \cdot x_{i,j} \leq T_{cj} \cdot y_j \quad \forall j \in J \tag{3.1}$$

$$\sum_{i=1}^{n} \rho_i^m \cdot x_{i,j} \leq T_{mj} \cdot y_j \quad \forall j \in J \tag{3.2}$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \forall i \in I \tag{3.3}$$

$$y_j, x_{i,j} \in 0, 1 \quad \forall j \in J \quad \text{and} \quad \forall i \in I \tag{3.4}$$

Constraints 3.1 and 3.2 guarantees that the capacity threshold of each server is not exceeded. Moreover, constraint 3.3 ensures that a VM is placed in exactly one server. Finally, constraint 3.4 represent the domain of variables $x_{i,j}$ and y_j.

3.3 CSO for Server Consolidation

This section discusses the development of the cuckoo search optimization (CSO) algorithm for solving the server consolidation problem in datacenters. The CSO algorithm employs a similar chromosome representation as the GGA, where genes represent groups (servers) instead of individual items (VMs). An illustration of such encoding is: A = {4,5,6}, B = {1,2,7}, C = {3,8}, D = {0,9,10}. Where A,B,C and D represent servers and numbers 0-10 represent VMs. Thus, this nest (chromosome) can be represented as: {4,5,6} {1,2,7} {3,8} {0,9,10}. The CSO algorithm for virtual machine placement is shown in Algorithm 6.

3.3.1 Algorthm Details

The algorithm begins by setting initial parameters such as fraction of population in the bottom nests P_a and the maximum number of iterations $MAXiter$. The initial population of size S is generated by initially obtaining S random permutations of the VM requests. Subsequently, for each of these random permutations, the FFD heuristic is run to obtain a placement solution. Thus, at the end we are able to obtain S different placement solutions to serve as the initial population. Next, the fitness of each nest is found by using the fitness measure described in Section 3.3.2. Subsequently, a procedure (line 6) commences and is repeated $MAXiter$ times. This procedure starts with the partitioning of the population into top and bottom nests. This partitioning is done by first obtaining the 75^{th} percentile of the population fitness. The first 75% of the population that have fitness less than or equal to the 75^{th} percentile form members of the bottom nests, and the remaining nests are selected as members of the top nest. This ranking method is used in order to avoid the computationally expensive sorting used in the modified cuckoo search algorithm [5]. Subsequently, for each nest belonging to the bottom nest (line 8-12) we generate a new nest using the $Perturb\_1$ function (described in section 3.3.3). The new nest generated is made to replace the old bottom nest. The population is then partitioned again into top and bottom nests.

Afterwards, the algorithm moves to the top nests procedure (line 14-21). For each top nest, a new nest X_k is generated from the current top nest by using the $Perturb\_2$ function (explained in 3.3.4). Subsequently, a random nest X_l is

picked from the entire population. If the fitness of the newly created nest (X_k) is better than that of X_l, it replaces X_l in the population. After the top nest procedure, the best nest found so far is stored and the outer-loop keeps repeating until $MAXiter$ is reached.

3.3.2 Fitness Evaluation

Sadiq *et al.* [15] proposed a fitness measure which is item (VM) based rather than the group (server) based proposed by Falkenauer [11] and Spieksma [16]. This fitness measure is intuitively speaking and considers the fitness of a packed VM as function of how well it utilizes the space remaining[1] in the server. A VM that fills this remaining space well receives a high fitness while one that fails to fill the space well gets a lower fitness. Equation 3.5 shows this fitness measure. The numerator denotes the VM in question, while the denominator represents the remaining CPU and memory capacity of the server assuming the VM is removed from the server. It should be noted that in the denominator of Equation 3.5, variable k cannot take the value of i.

$$\frac{\rho_i^c + \rho_i^m}{\left(T_c - \sum_{k=1}^{n} \rho_k^c\right) + \left(T_m - \sum_{k=1}^{n} \rho_k^m\right)} \tag{3.5}$$

As an illustration, consider the placement of three VMs on a server shown in Figures 3.1a and 3.1b with thresholds T_c and T_m set as 90%. In the case of Figure

[1]That is the residual space within the server assuming that the particular VM is removed from the server

Algorithm 6 Modified CSO algorithm

$P_a \leftarrow 0.75$ {P_a is the fraction of population belonging to bottom nests}
Set $MAXiter$ {$MAXiter$ is the maximum number of iteration}
Generation number $G \leftarrow 1$
Initialize the population
Calculate the fitness of each nest
for $G = 1$ **to** $MAXiter$ **do**
    Partition the population into top and bottom nests

---

    for all X_i such that X_i is in bottom nests **do**
        $X_j \leftarrow Perturb\_1(X_i)$
        $X_i \leftarrow X_j$
        $f(X_i) \leftarrow f(X_j)$
    end for
    Partition the population again into top and bottom nests

---

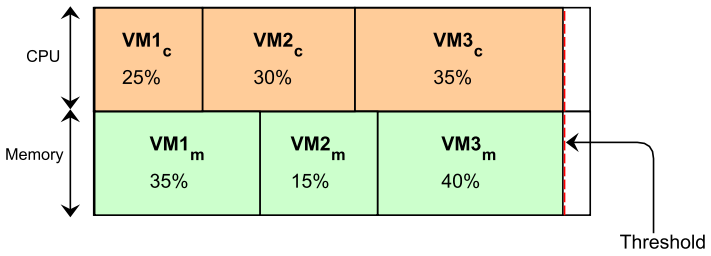
    for all X_i such that X_i is in top nests **do**
        $X_k \leftarrow Perturb\_2(X_i)$
        Select a random nest X_l from the entire population
        if $f(X_k) > f(X_l)$ **then**
            $X_l \leftarrow X_k$
            $f(X_l) \leftarrow f(X_k)$
        end if
    end for

    Store the best packing seen so far and its fitness
end for

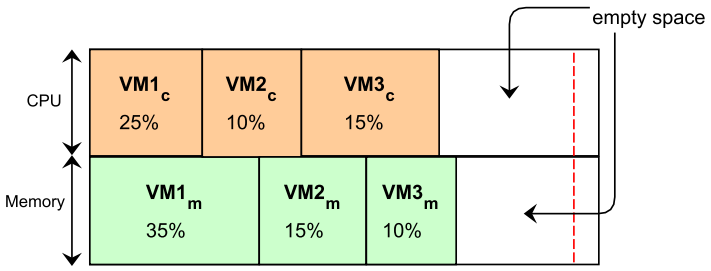
3.1a, from Equation 3.5 the fitness of VM1 is 1 and so is the finesses of the other two VMs since they completely fill the server's capacity. On the other hand, in Figure 3.1b, from Equation 3.5 the fitness of VM1 is computed as 0.46. This low fitness resulted because in Figure 3.1b VM1 does not utilize the remaining space within the server well. Thus Equation 3.5 is able to completely model how well VMs are placed within a server. Subsequently, the overall fitness of a nest (placement solution) is taken to be the average fitness of all VMs contained in the placement solution.

3.3.3 Perturb_1 Function

The CSO *perturb_1* function is executed in lines 9 and 17 of the CSO algorithm. The function receives a nest as input, it then generates a number x from a lévy distribution [4]: $x = (1 - u)^{-1/\alpha}$, where u is a uniform random variable in the range $[0, 1]$ and $\alpha = G^{1/6}$ with G as generation number. Subsequently, x number of servers are deleted from the nest. However due to this deletion, VMs contained in the deleted servers are missing from the nest and need to be reinserted. In order to avoid the scenario where VMs that were removed from the same server still remain in the same server after reinsertion, the missing VM list is randomly reshuffled. Finally, the sorted VMs are reinserted into the solution by using the first-fit (FF) heuristic and the function returns the completed solution.



(a) Perfect placement of VMs on a server.



(b) Placement of VMs on a server with wasted resources.

Figure 3.1: Illustrative example of VM placement.

3.3.4 Perturb_2 Function

The *perturb_2* function is executed in line 15 of the CSO algorithm within the top nests. The function receives a single nest as input. It then finds the fitness of each server within the nest. The fitness of each server is estimated as the average fitness of the VMs it hosts (calculated from Equation 3.5). The servers are then partitioned into top and bottom groups based on their fitness. Experimental testing has shown that by selecting the best 75% of servers as members of the top group and the remaining 25% as bottom group, better results are obtained. In order to avoid the sorting of servers within the nest, the partitioning process adopted is similar to that of partitioning the population into top and bottom nest explained in Section 3.3.1. Subsequently, the servers belonging to the bottom group are deleted from the nest. These deleted servers are then sorted according to one of the sorting methods of Maruyama *et al.* [8]. They are then replaced into the nest by the using the first-fit (FF) heuristic. The final resulting nest is then returned by the function.

3.3.5 Dataset

In order to generate the dataset, all that is needed is to develop sequences of random CPU and memory requests for VMs in an experiment that has several correlations. However as pointed out by Ajiro *et al.* [9] there is no available standard method for generating such sequences. Thus the method adopted is to use the probability P that both the CPU and memory utilization of a server would

be equal to or greater than some set reference values or that both utilizations would be less than the reference values. The following pseudocode is used to generate such instances of CPU and memory utilization requests.

for $i = 1$ **to** n

$\rho_i^c \leftarrow$ **rand**$(2R_c)$

$\rho_i^m \leftarrow$ **rand**(R_m)

$r =$ **rand**(1.0)

if $(r < P$ **and** $\rho_i^c \geq R_c)$ **or** $(r \geq P$ **and** $\rho_i^c < R_c)$

$\rho_i^m \leftarrow \rho_i^m + R_m$

end if

end for

In the procedure above, the function rand(x) returns random numbers that are uniformly distributed in the range [0, x). R_c and R_m represents the reference CPU and memory utilization respectively. These reference values are set as: $R_c = R_m = 25\%$ and $R_c = Rm = 45\%$. P is a probability that is used to control the correlations between memory utilizations and that of CPU. The value of P is set to: 0.0, 0.25, 0.50, 0.75 and 1.0 respectively. These values corresponds to strong negative, weak-negative, no, weak positive and strong positive correlations. Random VM requests are generated while incrementing the values of probability P. Thus for each reference value, 5 sets of VM requests are generated with each set representing different correlations.

3.3.6 Experimental Results

In this section we discuss the experimental findings. All algorithms are implemented in MATLAB R2012b (8.0.0.783) running on Windows 7 Ultimate 64-bit (6.1, build 7601) with 200GB RAM and Intel(R) Xeon(R) CPU X5690 3.5GHz (24CPUs). Datasets for experimentations are generated by using the procedure explained in Section 3.3.5. VM requests are produced for the case where the number of VM requests (n) is set to 200 and the case where it is set to 500 respectively. Thus, for each of the two cases ($n = 200$ and $n = 500$) there are 5 VM request sets, with each representing different correlations between the CPU and memory dimension of VM requests. For each correlation value we generated 100 instances of VM requests e.g. for the settings $n = 200$, $R_c = R_m = 25\%$ and $P = 0$ (strong negative correlation) we have 100 instances of 200 VM requests. The same is repeated for other settings. The CSO, RGGA [17] and GGA [11] algorithms are used to solve the placement of each of these 100 instances of VMs into physical servers and the average results are reported in Table 3.1. The population size and generation number of the CSO, RGGA [17] and GGA [11] algorithms is set to 25 and 100 respectively. The RGGA implemented is the steady state genetic algorithm [18] with crossover rate and mutation rates of 0.8 and 0.1 respectively as proposed by Wilcox *et al.* [17]. Similarly, the GGA implemented is a generational genetic algorithm with crossover rate of 0.8 and mutation rate of 0.1. Since the GGA fitness measure was designed to solve only single dimensional bin packing problem, the GGA implemented uses the RGGA fitness measure which

is a modification of the GGA fitness for solving multidimensional bin packing problems.

Table 3.1 compares the performance of the proposed CSO algorithm with that of RGGA [17] and GGA [11]. The performance measures are: The average number of physical machines used for placement(m), average consolidation ration (m/LB) and execution time in seconds. LB is the theoretical lower bound on the number of servers that can be used for placement given as:

$LB = \max\left\{\left\lceil\left(\sum_{i=1}^{n}\rho_i^c\right)/T_c\right\rceil, \left\lceil\left(\sum_{i=1}^{n}\rho_i^m\right)/T_m\right\rceil\right\}$. Thus, as the number of physical machines used for placement (m) approaches the theoretical lower bound (LB) the server consolidation ratio (m/LB) converges to a value of 1. From Table 3.1 we observe the following:

- For both cases i.e. where $n = 200$ and $n = 500$, the CSO algorithm outperforms the RGGA and GGA in terms of both average number of machines used for placement (m) and average server consolidation ratio (m/LB). Thus we can conclude that the CSO algorithm is able to find placement solutions with lesser number of machines than the RGGA and GGA.

- Comparing the performance of RGGA and the GGA algorithms, we can see that in the case where $n = 200$, the GGA outperforms the RGGA. In contrast for the case where $n = 500$, the RGGA performs better than the GGA in 7 out of the 10 cases reported.

- In terms of execution time, the RGGA has the fastest runtime because it is a steady state genetic algorithm. It is followed by the proposed CSO

algorithm and the GGA is the worst in terms of execution time.

- As correlation increases (from strong negative to strong positive), the number of machine needed for placement decreases. This is because when there is a negative correlation between the CPU and memory request of VMs, the VM placement generally results in plenty of wastage in each server and as a result a large number of servers are needed to accommodate the VMs.

- It can be observed that more number of servers(m) are needed for placement when the reference values R_c and R_m are set to 45% than when they are set to 25%. This is because in first scenario, the VM utilization requests are in the range [0,90%) while in the second case the range is [0,50%). Thus due to the VM sizes in the first case($R_c = R_m = 45\%$) more number of server are needed for placement.

- It is also obvious that at the same correlation and same reference value more servers are needed for placement in the case where $n = 200$ than the case where $n = 500$. This is because the number of servers needed for placement is proportional to the number of VM requests.

In the plots of Figures 3.2-3.4 we aim to compare the convergence rate of the proposed CSO algorithm with that of the RGGA and GGA algorithms. In this experiment we selected one of the 100 instances of case where $R_c = R_m = 45\%$, $n = 500$, $P = 0$ (strong negative correlation: Figure 3.2, $P = 3$ (zero correlation): Figure 3.3 and $P = 5$ (strong positive correlation): Figure 3.4 respectively. For

Table 3.1: Comparison of CSO with other VM-Placement Techniques

			n = 200			n = 500		
Reference value	**Corr.**	**Algorithm**	**m**	**m/LB**	**Time(s)**	**m**	**m/LB**	**Time(s)**
$R_c = R_m = 25\%$	strong -ve	CSO	59.40	1.03	5.66	145.86	1.03	22.68
		RGGA	62.26	1.08	2.03	153.63	1.08	9.58
		GGA	61.32	1.06	5.94	152.79	1.08	25.03
	weak -ve	CSO	58.98	1.02	5.64	144.31	1.02	22.64
		RGGA	60.91	1.05	1.95	149.22	1.05	8.99
		GGA	60.44	1.04	5.80	149.27	1.05	24.53
	zero	CSO	58.75	1.02	5.80	143.76	1.01	23.21
		RGGA	60.44	1.05	1.96	147.67	1.04	8.85
		GGA	59.84	1.04	5.72	148.00	1.04	25.05
	weak +ve	CSO	57.96	1.02	5.80	143.19	1.01	23.95
		RGGA	59.53	1.04	1.91	146.15	1.03	8.82
		GGA	59.08	1.04	5.65	146.75	1.04	25.39
	strong +ve	CSO	57.74	1.01	5.83	142.19	1.01	24.30
		RGGA	58.87	1.03	1.88	144.74	1.03	8.69
		GGA	58.59	1.03	5.61	145.30	1.03	25.15
$R_c = R_p = 45\%$	strong -ve	CSO	121.01	1.17	9.56	290.75	1.14	42.07
		RGGA	123.79	1.20	3.82	299.30	1.17	18.41
		GGA	122.34	1.18	10.02	297.89	1.17	50.01
	weak -ve	CSO	118.48	1.15	9.25	286.27	1.12	41.08
		RGGA	121.40	1.17	3.72	294.50	1.15	18.00
		GGA	120.05	1.16	9.82	293.69	1.15	48.71
	zero	CSO	116.07	1.13	8.99	278.96	1.10	39.76
		RGGA	118.96	1.15	3.65	287.73	1.13	17.66
		GGA	117.91	1.14	9.62	287.75	1.13	47.59
	weak +ve	CSO	114.11	1.11	8.90	272.39	1.08	38.72
		RGGA	117.05	1.14	3.52	280.58	1.11	17.08
		GGA	116.05	1.13	9.48	281.35	1.11	46.33
	strong +ve	CSO	109.64	1.07	8.66	267.71	1.06	37.93
		RGGA	112.32	1.10	3.38	274.52	1.09	16.59
		GGA	111.43	1.09	9.16	275.27	1.09	45.36

each experiment, the same initial population was fed into the algorithms and each of the algorithms is run 10 times on the same initial population and then the best traces obtained are plotted. From the plots in Figures 3.2-3.4 we can observe that after few iterations the CSO has faster convergence rate than the RGGA and GGA algorithms. Moreover, looking at the number of servers used for placement at the end of the 100^{th} iteration, we can see that the CSO clearly outperforms the

RGGA and GGA algorithms.

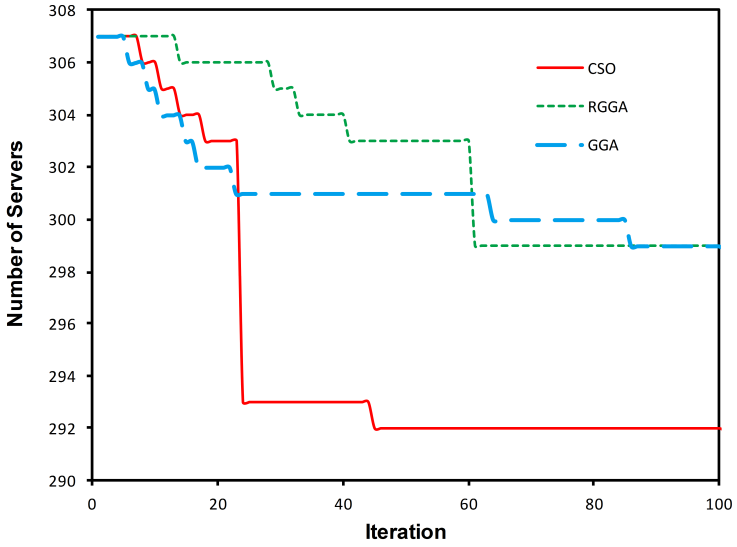


Figure 3.2: Convergence of CSO with other Techniques (strong neg. corr.)

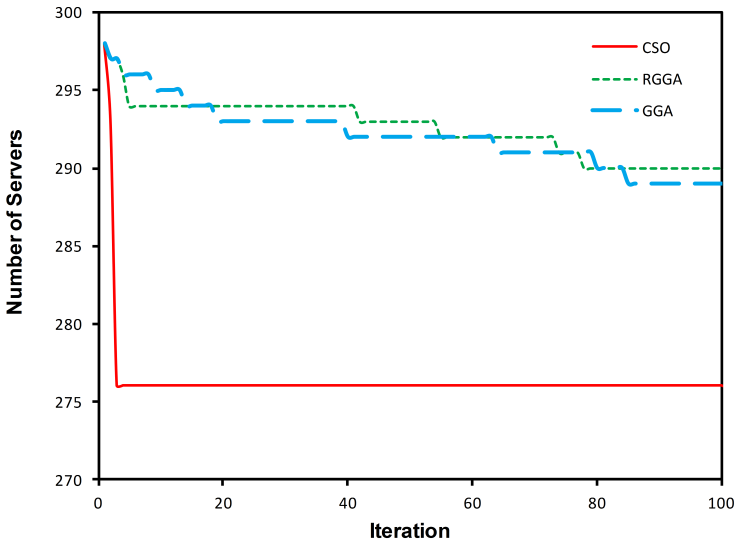


Figure 3.3: Convergence of CSO with other Techniques (zero corr.)

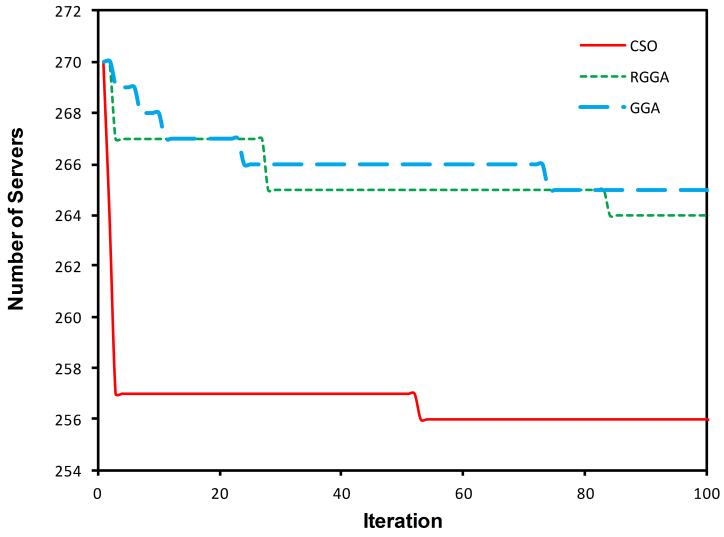


Figure 3.4: Convergence of CSO with other Techniques (strong post. corr.)

3.4 Chapter Summary

In this chapter, we employed the cuckoo search optimization (CSO) algorithm to solve the server consolidation problem in datacenters. Results we obtain show that the CSO outperforms the Reordered Grouping Genetic Algorithm (RGGA) and Grouping Genetic Algorithm (GGA).In the next chapter we look into the second part of this thesis, where we employ the use of multi-objective CSO to simultaneously optimize the datacenter for reduced power consumption and resource wastage.

CHAPTER 4

MULTI-OBJECTIVE CSO FOR VM-PLACEMENT

In this chapter we discuss the design of the cuckoo search optimization (CSO) algorithm for virtual machine placement in cloud computing environments. The designed CSO is a multi-objective cuckoo search algorithm that maps VMs to physical machines while simultaneously minimizing the power consumption and resource wastage of the datacenter. Moreover, we employ the use of And-Like-Fuzzy Aggregation (AFA) function to evaluate the fitness of solutions in the population pool.

The performance of the CSO is compared with that of the reordered grouping genetic algorithm (RGGA) and grouping genetic algorithm (GGA) In all cases the CSO was found to outperform the other VM placement methods.

4.1 Problem Description

This section discusses the optimization equations of the VM placement problem In this study, we consider the datacenter to be fully virtualized such that all applications are running on virtual machines. CPU and memory dimensions are used to characterize a VM and its host server. If more than one VM is running on a server node, the CPU and memory utilization of the server is estimated as the sum of CPU utilizations of the VMs and memory utilizations of the VMs respectively. However in order to prevent the server from reaching 100% resource utilization, a threshold is set as an upper bound on resource utilization on each server. The main reason for this is that allowing the server to reach 100% utilization can cause huge performance degradation.

4.1.1 Resource Wastage Modeling

Different VM placement solutions often result in varied resource wastages on each server. Thus to fully utilize multidimensional resources, potential cost of wasted resources is computed using the following equation [10]:

$$W_j = \frac{\mid L_j^m - L_j^c \mid + \varepsilon}{U_j^m + U_j^c} \tag{4.1}$$

Where: W_j represents the resource wastage of the j^{th} server, U_j^m and U_j^c denotes the normalized memory and CPU resource usage (i.e., the ratio of used resource to total available resource) respectively. L_j^m and L_j^c represents the normalized remaining memory and CPU resource respectively, while ε is a constant

with value set as 0.0001. This constant (ε) is added to prevent the case where the resource wastage of a server is returned as zero. The main idea of Equation 4.1 is to make efficient utilization of resources in all dimensions and balance the remaining resources on each server along different dimensions. Thus, from Equation 4.1, a server with almost equal CPU and memory utilization will have a low resource wastage. This means that it has a high probability of receiving additional VMs.In contrast, when the difference between the CPU and memory utilization of a server is large, the resource utilization becomes higher. Which means that such a server is less likely to accept new incoming VMs.

4.1.2 Power Consumption Modeling

Fan *et al.* [19] proposed a method for accurately determining the power consumption of a server from the linear relationship between power consumed and the utilization of CPU. This research has been further verified by Gao *et al.* [10] in experiments they conducted on a Dell server. In order to save energy, idle servers[1] are usually turned off. Thus their power consumption in idle state is not part of the total energy consumed by the CPU. The power consumed by the j^{th} server can be defined by Equation 4.2 [10].

$$P_j = \begin{cases} \left[(\bar{p}_{busy} - \bar{p}_{idle}) \times U_j^c\right] + \bar{p}_{idle}, & \text{if } U_j^c > 0. \\ 0, & \text{elsewhere.} \end{cases} \tag{4.2}$$

Since we are considering physical servers to be homogeneous, $\bar{p}_{busy}$ and $\bar{p}_{idle}$

are the average power consumption of a server when it is fully utilized and when it is idle, respectively. Gao *et al.* [10] conducted experiments on a Dell server and came up with $\bar{p}_{busy} = 215\,Watts$ and $\bar{p}_{idle} = 162\,Watts$.

4.1.3 Optimization Equations

In this section the VM placement optimization problem is formulated. Given n number of VMs to be placed on m servers. Additionally, assuming that no VM has more resource requests that can be provided by a single server. Let ρ_i^c denote CPU demand of each VM, ρ_i^m be the memory demand of each VM and T_{cj}, T_{mj} be the threshold of CPU and memory utilization of each server respectively. Furthermore, let x_{ij} be a binary value indicating whether VM_i is assigned to server j. Moreover, we take y_j another binary value to indicate whether a server is in use or not. Since the aim is to minimize both power consumption and resource wastage, the overall placement problem can be formulated as [10]:

$$Minimize \sum_{j=1}^{m} P_j = \sum_{j=1}^{m} \left\{ y_j \times \left[\left(\bar{p}_j^{busy} - \bar{p}_j^{idle} \right) \times \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^c) + \bar{p}_j^{idle} \right] \right\}$$

$$Minimize \sum_{j=1}^{m} W_j = \sum_{j=1}^{m} \left\{ y_j \times \frac{\left| \left(T_{cj} - \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^c) \right) - \left(T_{mi} - \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^m) \right) \right| + \varepsilon}{\sum_{i=1}^{n} (x_{ij} \cdot \rho_i^c) + \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^m)} \right\}$$

Subject to:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in I \tag{4.3}$$

$$\sum_{i=1}^{n} \rho_i^c \cdot x_{ij} \leq T_{cj} \cdot y_j \quad \forall j \in J \tag{4.4}$$

$$\sum_{i=1}^{n} \rho_i^m \cdot x_{ij} \leq T_{mj} \cdot y_j \quad \forall j \in J \tag{4.5}$$

$$y_j, x_{ij} \in 0, 1 \quad \forall j \in J \quad \text{and} \quad \forall i \in I \tag{4.6}$$

Constraint 4.3 checks that a VM is not placed in more than one server. In addition, constraints 4.4 and 4.5 are used to model the capacity threshold of the server. Moreover, constraint 4.6 represent the domain of variables y_j and x_{ij}.

4.2 Multi-objective CSO Algorithm with Fuzzy Evaluation

The multiobjective CSO algorithm for VM placement is similar to the CSO algorithm for server consolidation discussed in Chapter 3 . Both algorithms have the same encoding and *Perturb_1* functions. However, they differ in two main areas, which are the fitness evaluation method and *Perturb_2* function. While the CSO for server consolidation uses the fitness evaluation method detailed in Section 3.3.2, the multiobjective CSO uses a fuzzy evaluation method (explained in Section 4.2.1) to combine both the power consumption and resource wastage

objectives into one objective which we aim to maximize. Moreover, the *Perturb_2* function of the CSO for server consolidation (explained in Section 3.3.4) and *Perturb_2* of the multiobjective CSO are the same in all aspects, except in the method used in finding the fitness of a server. In this case, the function for finding server fitness operates as follows: Firstly, the function receives a server as input, say server x. Upper and lower bounds are then defined for server power and resource wastage. The lower bound on power is estimated as $\bar{P}_{idle} = 162W$ i.e. Power consumed when the server is almost idle. The upper bound on power is taken to be the power consumed a when the server is fully loaded i.e. when the CPU utilization is 90% (threshold). In contrast, the resource wastage lower bound is taken to be resource wastage generated when the server is fully loaded i.e. both CPU and memory utilization of the server is 90%. The upper bound on resource wastage is taken to be the case where the resource utilization of a server is skewed i.e. the case in which one dimension is fully utilized while the other is almost zero e.g. CPU utilization is 90% and memory utilization is 0%. The wastage generated from such a server is taken as upper bound on server resource wastage. The function then computes the actual power and resource wastage of server x. Equation 4.12 is then used to find the membership of server x in fuzzy set of {lp} and {sw}, the average of these membership values is then taken to be the fitness of server x.

4.2.1 Fuzzy Fitness Evaluation

Fitness Measure

The fitness of a particular nest or placement solution is obtained by fuzzy logic [20]. Fuzzy logic enables different criteria to be mapped into linguistic values which characterize the designer's level of satisfaction with the numerical values of the objectives [21]. These linguistic values operate over the interval $[0 \sim 1]$. The following fuzzy rule can be used to express the evaluation of a solution: **IF** solution x has low power consumption (lp), **AND** small resource wastage (sw) **THEN** it is a good solution. Thus, the solution with the best quality is the one with highest membership in the fuzzy sets of $\{lp, sw\}$. We use the And-like-Fuzzy-Aggregation (AFA) fuzzy rule proposed by Khan *et al.* [21] to obtain the fitness of solutions. However, in order to obtain the fitness of a solution we have to find its membership in fuzzy set of $\{lp\}$ and $\{sw\}$ respectively. This membership can be obtained by first defining upper and lower bound for power consumption and resource wastage.

Power consumption bounds

Lower bound: The lower bound on power is the power consumed by a placement solution consisting of minimum number of servers that are required to pack VMs. This theoretical lower bound on number of servers M_{min} is computed using Equation 4.7. Thus, assuming all these servers are busy, the lower bound on power is given by Equation 4.8. In Equation 4.8, the first term denotes the total power

consumed by other server peripherals, while the second term represents the total power consumed by the CPUs of M_{min} servers.

$$M_{min} = \max\left\{\left\lceil\left(\sum_{i=1}^{n}\rho_i^c\right)/T_c\right\rceil, \left\lceil\left(\sum_{i=1}^{n}\rho_i^m\right)/T_m\right\rceil\right\} \tag{4.7}$$

$$Power_{lower} = M_{min}\bar{P}_{idle} + \bar{P}_{cpu}\sum_{i=1}^{n}(\rho_i^c) \tag{4.8}$$

Upper bound: We assume the upper bound on power consumption to be the real power consumed by the datacenter when the maximum possible number of servers is used for placement i.e. when we have one VM per server denoted by $M_{max} \equiv$ number of VMs. Thus, the upper bound on power is:

$$Power_{upper} = M_{max}\bar{P}_{idle} + \bar{P}_{cpu}\sum_{i=1}^{n}(\rho_i^c) \tag{4.9}$$

$\bar{P}_{cpu} = \bar{P}_{busy} - \bar{P}_{idle}$ is the average power consumed by the CPU and ρ_i^c is the CPU utilization of VMs. T_c and T_m represent the CPU and memory utilization capacity of a server.

wastage Bounds

Lower bound: In the case of resource wastage, we assume the lower bound to be the wastage calculated when we assume that there exist a large single server that can accommodate all VMs. The wastage that is generated by the placement of VMs in this large server is the lower bound on resource wastage and is given

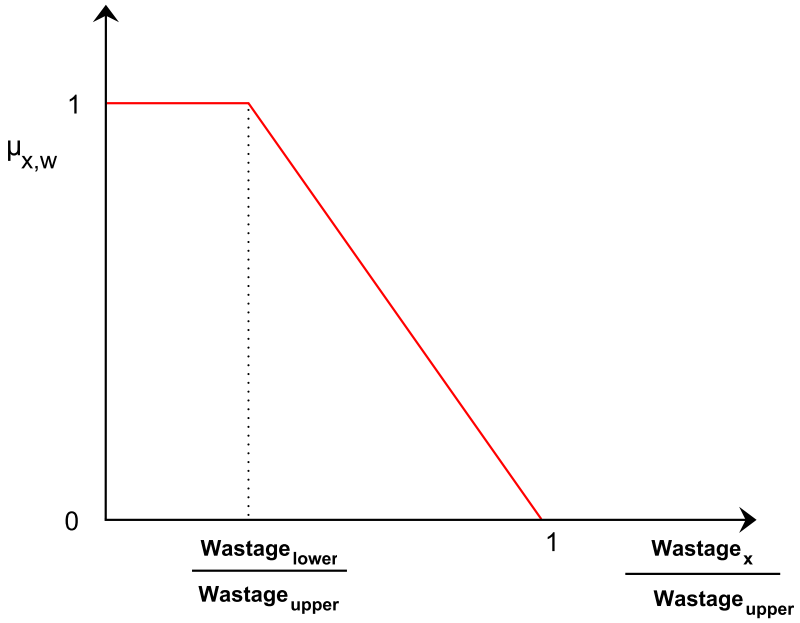
by:

$$Wastage_{lower} = \frac{|\sum_{i=1}^{n}(\rho_i^c) - \sum_{i=1}^{n}(\rho_i^m)| + \varepsilon}{\sum_{i=1}^{n}(\rho_i^c) + \sum_{i=1}^{n}(\rho_i^m)} \tag{4.10}$$

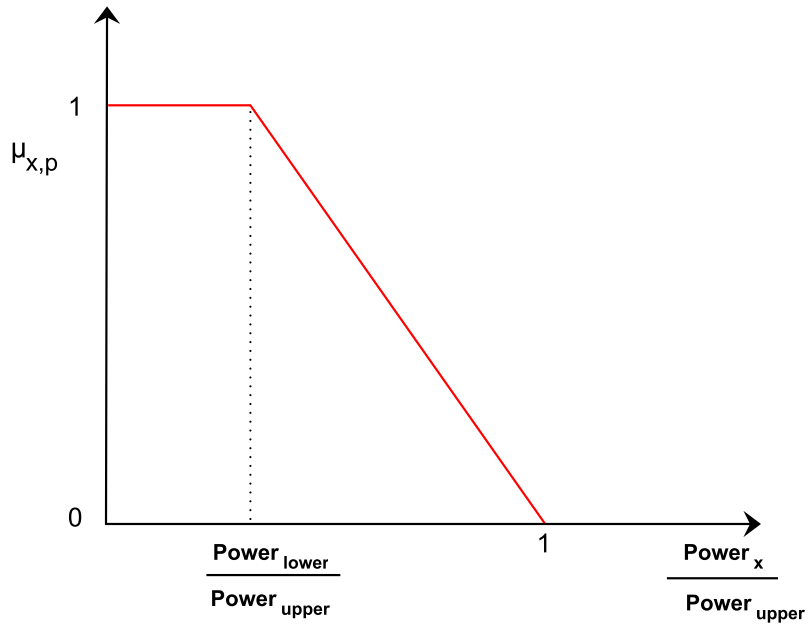
Upper bound: The upper bound on resource wastage is the wastage obtained from the worst placement of VMs i.e.,one VM per server. This wastage is given by:

$$Wastage_{upper} = \sum_{i=1}^{n}\left\{\frac{|\rho_i^c - \rho_i^m| + \varepsilon}{\rho_i^c + \rho_i^m}\right\} \tag{4.11}$$

Subsequently, from these computed bounds, we are able to find the membership of a solution say x in the fuzzy set of $\{lp, sw\}$. The *Fitness* function receives inputs of upper and lower bounds on power and resource wastage. Moreover, it also receives input of the resource wastage ($Wastage_x$) and power consumption ($Power_x$) of solution x computed from Equations 4.1 and 4.2 respectively. From these inputs, the function is able to compute the membership of solution x in the set $\{sw\}$ and set $\{lp\}$ given by $\mu_{x,w}$, $\mu_{x,p}$ in Equation 4.12.



(a) Plot of the Membership of Solution x in the Set {sw}.



(b) Plot of the Membership of Solution x in the Set {lp}.

Figure 4.1: Normalized membership function of solution x

$$\mu_{x,w} = \frac{Wastage_{upper} - Wastage_x}{Wastage_{upper} - Wastage_{lower}} \tag{4.12a}$$

$$\mu_{x,p} = \frac{Power_{upper} - Power_x}{Power_{upper} - Power_{lower}} \tag{4.12b}$$

By employing the equations of Khan *et al.* [21], we compute the weights of each objective.

Equation 4.13 is used to computer such weights.

$$\bar{w}_{x,w} = \frac{\bar{\mu}_{x,w}}{\bar{\mu}_{x,w} + \bar{\mu}_{x,p}} \qquad \bar{w}_{x,p} = \frac{\bar{\mu}_{x,p}}{\bar{\mu}_{x,p} + \bar{\mu}_{x,w}} \tag{4.13}$$

where: $\bar{\mu}_{x,w} = 1 - \mu_{x,w}$ and $\bar{\mu}_{x,p} = 1 - \mu_{x,p}$

Figure 4.1 shows the graph of the normalized membership function of solution x in the set {sw} and {lp} respectively.

From these weights we find the complement of the overall membership of solution x in the fuzzy set $\{lp, sw\}$ as:

$$\bar{\mu}_x = (\bar{w}_{x,w})(\bar{\mu}_{x,w}) + (\bar{w}_{x,p})(\bar{\mu}_{x,p}) \tag{4.14}$$

Finally, we are able to obtain the fitness evaluation of solution x from Equation 4.14.

$$\mu_x = 1 - \bar{\mu}_x \tag{4.15}$$

As an illustration of how the overall membership of a particular solution is found, take for example a nest x with overall power consumption and resource wastage calculated from Equations 4.1 and 4.2 as ($Power_x = 300$) and ($Wastage_x = 65$) respectively. To find the membership of such nest, we start by computing upper and lower bounds from Equations 4.8 through 4.11. Let us assume that these bounds were found to be; $Power_{lower} = 200$, $Power_{upper} = 650$, $Wastage_{lower} = 50$ and $Wastage_{upper} = 100$. Thus, from Equation 4.12 we find that for nest x, $\mu_{x,w} = 0.7$ and $\mu_{x,p} = 0.78$. Subsequently, from Equation 4.13 we find the weights to be $\bar{w}_{x,w} = 0.58$ and $\bar{w}_{x,p} = 0.423$. Consequently, from Equation 4.14 we find that $\bar{\mu}_x$ is 0.27. Finally, from Equation 4.15 we obtain $\mu_x = 0.73$ which is the overall membership value of nest x in fuzzy set of good solutions.

4.3 Experimental Results

In this section we present our experimental results on the employment of the multiobjective CSO algorithm to solve the VM placement problem of datacenters. The experimental platform and parameters used are similar to that discussed in Section 3.3.6. Table 4.1 shows comparison results of the multiobjective CSO, RGGA and GGA algorithms. The measures for comparison are: the average power consumption (Power), average resource wastage (RW), average fuzzy fitness (FF) and CPU times (Times). From Table 4.1 the following points can be observed:

- In terms of power consumption, in the scenario where $n = 200$ the CSO performs better than both the RGGA and GGA algorithms in 8 out of the

10 cases reported. In contrast, in the situation where $n = 500$, the CSO outperforms the RGGA and GGA in 9 out of the 10 cases. Considering average resource wastage, the CSO algorithm outperforms the RGGA and GGA algorithms in all cases. In terms of fuzzy fitness (FF), the CSO performs better than both the RGGA and GGA algorithms in all cases.

- As correlation increases i.e. from strong negative to strong positive, the average power and average resource wastage decreases for all algorithms. This is because as correlation increases less number of servers are needed for VM placement and thus resulting in low power consumption and resource wastage.

- Comparing run times, the RGGA has the least run time followed by the CSO and then the GGA.

- More power consumption and resources wastage are generated for the case where $n = 500$ than the case where $n = 200$. This is because with more number of VMs, more servers are needed for placement and as a result more power and resource wastage is produced.

Figure 4.2 shows how the fuzzy fitness and average fuzzy fitness of solutions (nests) improve with iteration. Without the loss of generality, the case reported in where $n = 200$, $R_c = R_m = 25\%$ and $P = 0$ (strong negative correlation). However, similar result is obtained from other cases. From Figure 4.2, we can see that the best fuzzy fitness curve increases smoothly with iteration. While in the case of

Table 4.1: Comparison of Multi-CSO with other VM-Placement Techniques

			n = 200				n = 500			
Reference value	**Corr.**	**Algorithm**	**Power(W)**	**Wastage**	**FF ($\times 10^3$)**	**Time (s)**	**Power(W)**	**Wastage**	**FF ($\times 10^3$)**	**Time (s)**
$R_p = R_m = 25\%$	strong -ve	CSO	12697	3.60	965	4.12	31460	8.18	967	18.25
		RGGA	12739	5.14	956	2.03	31514	10.22	963	9.58
		GGA	12587	4.95	959	5.94	31378	10.08	964	25.03
	weak -ve	CSO	12462	3.00	970	3.96	30638	5.98	976	17.09
		RGGA	12528	4.39	959	1.95	30801	7.95	970	8.99
		GGA	12452	4.39	959	5.80	30809	7.84	970	24.53
	zero	CSO	12350	2.30	974	3.92	30388	4.77	979	16.93
		RGGA	12463	3.81	958	1.96	30556	6.71	971	8.85
		GGA	12366	3.41	963	5.72	30610	6.62	971	25.05
	weak +ve	CSO	12193	1.76	976	3.89	30194	4.19	978	16.80
		RGGA	12300	3.05	960	1.91	30304	5.77	970	8.82
		GGA	12227	2.76	964	5.65	30402	6.01	969	25.39
	strong +ve	CSO	12127	1.17	979	3.85	30012	2.83	981	16.54
		RGGA	12198	2.00	966	1.88	30096	3.91	974	8.69
		GGA	12152	1.76	970	5.61	30187	3.91	974	25.15
$R_m = R_p = 45\%$	strong -ve	CSO	24443	18.86	812	6.68	59281	38.58	848	34.97
		RGGA	24841	23.62	777	3.82	60391	48.11	816	18.41
		GGA	24607	23.63	784	10.02	60162	48.19	819	50.10
	weak -ve	CSO	23966	15.40	833	6.52	58564	32.71	861	31.80
		RGGA	24423	19.81	794	3.72	59656	40.09	831	18.00
		GGA	24204	20.17	797	9.82	59525	40.58	831	48.71
	zero	CSO	23626	12.83	844	6.35	57403	24.73	879	30.74
		RGGA	24049	16.39	806	3.65	58519	31.46	848	17.66
		GGA	23879	17.18	803	9.62	58522	32.77	844	47.59
	weak +ve	CSO	23336	9.62	861	6.30	56400	18.70	893	29.72
		RGGA	23770	12.81	820	3.52	57346	23.73	865	17.08
		GGA	23608	13.18	819	9.48	57471	25.56	857	46.33
	strong +ve	CSO	22624	5.58	898	6.11	55779	12.43	911	28.49
		RGGA	22985	7.76	862	3.38	56418	15.67	889	16.59
		GGA	22841	7.61	867	9.16	56540	16.58	883	45.36

the average fuzzy fitness curve, it can be seen that it rises and falls with iteration. This is because in the CSO algorithm (line 8-12), all bottom nests are replaced irrespective of their quality, thus the average quality of nests is bound to rise and

fall with iteration. This attribute is similar to hill climbing and it is one of the attractive features of the CSO algorithm that helps it escape the local minima of most optimization problems.

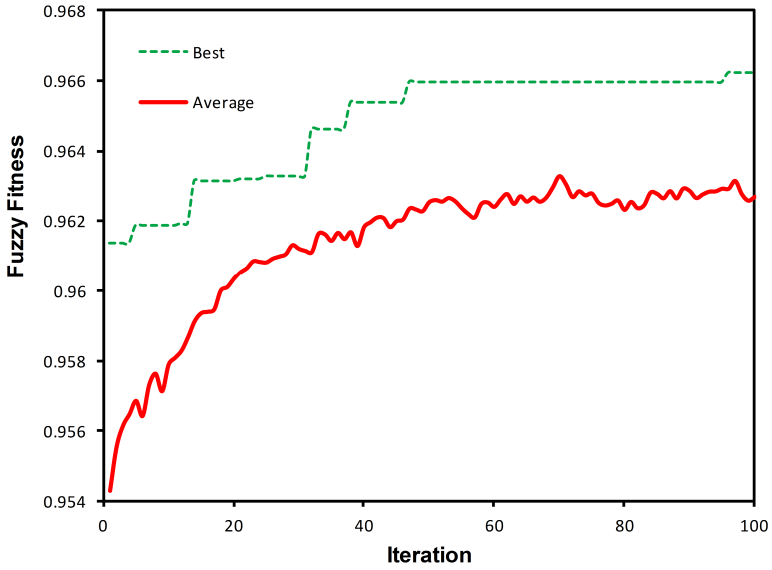


Figure 4.2: plot of best and average fuzzy fitness per iteration.

4.4 Chapter Summary

In this chapter we discuss the design of a cuckoo search optimization (CSO) algorithm for virtual machine placement in cloud computing environments. The designed CSO is a multi-objective cuckoo search algorithm that maps VMs to physical machines while simultaneously minimizing the power consumption and resource wastage of the datacenter. Moreover, we employ the use of And-Like-Fuzzy Aggregation (AFA) function to evaluate the fitness of solutions in the population pool.

The performance of the CSO is compared with that of the reordered grouping genetic algorithm (RGGA) and grouping genetic algorithm (GGA). In all cases the CSO was found to outperform the other VM placement methods.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

This chapter presents concluding remarks of this thesis research work. Moreover, the chapter highlights certain important areas in which this thesis research can be extended.

5.1 Conclusions

This thesis research involved the employment of cuckoo search optimization (CSO) algorithm, a novel nature-inspired metaheuristic algorithm to solve the virtual machine (VM) placement problem in datacenters. The VM placement problem is a variant of the vector bin-packing problem which is an NP-hard optimization problem. This research work is divided into two parts. In the first part the CSO algorithm is engineered to solve the server consolidation problem in datacenters. In the server consolidation problem it is required that virtual machines be placed

onto the datacenter while minimizing the number of physical servers used for placement. The performance of the CSO algorithm for server consolidation is compared with reordered grouping genetic algorithm (RGGA) and grouping genetic algorithm (GGA). Performance indices for comparison are: the number of physical machines used for placement (m) and server consolidation ratio (m/LB), where LB is the theoretical lower bound on the number of servers. From comparison results, it was found that the CSO algorithm was able to outperform other methods within a very competitive computational time. This clearly shows that the CSO algorithm for server consolidation is robust enough and can be used to efficiently solve the server consolidation problem of datacenters.

In the second part of this research work, the CSO algorithm is also employed to solve the virtual machine placement problem in datacenters. However, in this case, the aim of the CSO algorithm is to simultaneously minimize the power consumption and resource wastage of the datacenter. This kind of optimization is widely known as multi-objective optimization. In addition, the And-Like-Fuzzy-Aggregation multi-objective evaluation function is used to determine the fitness of solution in the population pool of the CSO. The And-Like-Fuzzy-Aggregation combines the power consumption and resource wastage into a single objective that needs to be maximized. The fuzzy fitnesses of solutions obtained with the CSO algorithm is compared with that obtained from reordered grouping genetic algorithm (RGGA) and grouping genetic algorithm (GGA) methods of VM placement. From obtained comparison results, it was found that the CSO was able

to obtain solutions with higher fuzzy evaluation fitness than that obtained from other methods.

The CSO algorithm for virtual machine placement can also be employed in solving the dynamic placement of VMs in datacenters. When a batch of VM requests are received by the datacenter, the CSO algorithm is used to resolve the placement problem in order to find a good placements for both the incoming and existing VMs. Moreover, because VM requests can time-out, it is recommended that after several weeks, the CSO algorithm should be used again to resolve the placement problem so that better placements can be found for existing VMs. However this resolving process should not be done frequently, due to the expensive nature of VM-migration. In the case that only few VMs arrive at the datacenter, it is better to use the FFD heuristic to place them.

5.2 Future Work

In this section we present possible extension to our research work.

5.2.1 Heterogeneous Datacenters

An interesting extension to this research work is the application of the CSO algorithm for virtual machine placement in heterogeneous datacenters. Physical servers in such datacenters are not identical. This poses a great challenge to the global controller. As stated earlier, the local controller maps application requests to resource utilization requests either by estimations or profiling. However, this

mapping is usually done for a particular type of server. Thus when the global controller attempts to place a VM on a different type of server, it has to re-estimate the resource utilization of the VM on the new type of server. This re-estimation by the global controller any time it attempts to place a VM on a different type of server can be a very challenging task. Nonetheless, given CSO's performance on the placement of VMs in homogeneous datacenters, it is projected that it will also outperform other methods in VM placements involving heterogeneous datacenters.

5.2.2 Hardware Validation Testing

Another interesting future direction for this work will be to test the CSO algorithm on real cloud environments so as to corroborate simulation result with real experimental results.

5.2.3 Testing Algorithm Parameters

Furthermore, another very important future area of study is to test how CSO algorithm parameters such as: the population size and number of iteration affects the performance of the CSO algorithm. This research will attempt to find ideal algorithm parameters for a particular size of VM placement problem.

5.2.4 Multidimensional Vector Bin-Packing

Although the CSO designed and implemented here is for VM placement and typical dimensions in such problems are: memory, CPU, and disk, it will be very

interesting to see the performance of the CSO in solving vector bin-packing problems that have many dimensions such as 5 or 6. This test will further prove the robustness of the CSO algorithm in solving other grouping problems.

5.3 Chapter Summary

This chapter summarises the research findings of this theses. Subsequently the chapter also discusses certain key areas in which this research can be improved.

REFERENCES

[1] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom).* IEEE, 2010, pp. 179–188.

[2] S. M. Sait and H. Youssef, *Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems.* IEEE Computer Society Press, 1999.

[3] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble." *HotPower*, vol. 8, pp. 2–2, 2008.

[4] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on.* IEEE, 2009, pp. 210–214.

[5] S. Walton, O. Hassan, K. Morgan, and M. Brown, "Modified cuckoo search: a new gradient free optimisation algorithm," *Chaos, Solitons & Fractals*, vol. 44, no. 9, pp. 710–718, 2011.

[6] I. Pavlyukevich, "Lévy flights, non-local search and simulated annealing," *Journal of Computational Physics*, vol. 226, no. 2, pp. 1830–1844, 2007.

[7] C. T. Brown, L. S. Liebovitch, and R. Glendon, "Lévy flights in dobe Ju/hoansi foraging patterns," *Human Ecology*, vol. 35, no. 1, pp. 129–138, 2007.

[8] K. Maruyama, S. Chang, and D. Tang, "A general packing algorithm for multidimensional resource requirements," *International Journal of Computer & Information Sciences*, vol. 6, no. 2, pp. 131–149, 1977.

[9] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Int. CMG Conference*, 2007, pp. 399–406.

[10] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.

[11] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

[12] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proceedings of the 2011 IEEE/ACM 12th Interna-*

tional Conference on Grid Computing. IEEE Computer Society, 2011, pp. 26–33.

[13] A. H. El-Maleh, S. M. Sait, and A. Bala, "Cuckoo search optimization in state assignment for area minimization of sequential circuits," in *Computers and Their Applications, 2014. CATA-2014. Accepted in. 29th International Conference on*, 2014.

[14] I. Fister Jr, X.-S. Yang, D. Fister, and I. Fister, "Cuckoo search: A brief literature review," in *Cuckoo search and firefly algorithm.* Springer, 2014, pp. 49–62.

[15] S. M. Sait and S. K. Shahid, "Engineering simulated evolution for virtual machine assignment," submitted To: Journal of Applied Research and Technology.

[16] F. C. Spieksma, "A branch-and-bound algorithm for the two-dimensional vector packing problem," *Computers & operations research*, vol. 21, no. 1, pp. 19–25, 1994.

[17] D. Wilcox, A. McNabb, and K. Seppi, "Solving virtual machine packing with a reordering grouping genetic algorithm," in *Evolutionary Computation (CEC), 2011 IEEE Congress on.* IEEE, 2011, pp. 362–369.

[18] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[19] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[20] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 18, no. 1, pp. 183–190, 1988.

[21] J. A. Khan and S. M. Sait, "Fuzzy aggregating functions for multiobjective vlsi placement," in *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, vol. 2. IEEE, 2002, pp. 831–836.

Vitae

- Name: Abubakar Bala.
- Nationality: Nigerian.
- Date of Birth: 23-03-1988.
- SEPT. 2012 - DEC. 2014: Master's Degree in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia.
- JAN. 2006 - MAR. 2011: Bachelor's degree in Computer Engineering from Bayero University Kano (BUK), Kano, Nigeria.
- SEPT. 1999 - JUNE 2005: Senior secondary school certificate from ST. Thomas' Secondary School, Kano Nigeria.
- SEPT. 1991 - JULY 1999: First school leaving certificate from Corona Primary School, Bukuru, Jos, Plateau State, Nigeria.
- JUNE 2011 - TO DATE : Graduate Assistant (GA) at Bayero University Kano (BUK) Kano, Nigeria.
- Email: *balaabubakar2006@yahoo.com*
- Permenant Address: No. 1, Yusuf Ali Link, Dan-Agundi Industrial Layout, P.O.BOX 10871, Sharada, Kano State, Nigeria

- Journal Publications:

1. **Bala, A.**, & Osais, Y. (2013). Modelling and simulation of DDOS Attack using SimEvents. International Journal of Scientific Research in Network Security and Communication, 5-14.
2. El-Maleh, A. H., Sait, S. M., & **Bala, A.** (2015). State Assignment for Area Minimization of Sequential Circuits Based on Cuckoo Search Optimization. Computers and Electrical Engineering (Under Review).
3. Sait, S. M., **Bala, A.**, & El-Maleh, A. H. (2015). Cuckoo Search Optimization Based Virtual Machine Placement in datacenters. (in Writing)